RaligNAtor
User's Manual

Fernando Meyer
Center for Bioinformatics
University of Hamburg
Bundesstr. 43, 20146 Hamburg, Germany

June 20, 2013

Contents

1 Introduction

RaligNAtor is a software package for fast approximate matching of RNA sequence-structure patterns. It searches sequence databases for occurrences of user-given patterns annotated with secondary structure. Its main features are:

- Implementations of new efficient user-selectable online and index-based matching algorithms.
- Matching computation based on a sequence-structure edit distance with a full set of edit operations on single bases and base pairs.
- Patterns can describe any (branching, non-crossing) RNA secondary structures. Sequence information can contain ambiguous IUPAC symbols.
- Search in DNA and RNA sequences possible due to flexible alphabet handling.
- Matching on forward and reverse complement strands.
- Customizable base pairing rules.
- Integrated fast algorithms for global and local chaining of matches.
- Output of results including matching positions, sequence-structure alignments, scores, etc.

For index-based matching, *RaligNAtor* uses a data structure based on the suffix array precomputed from the target sequence database. This precomputation is performed by the *sufconstruct* tool distributed with *RaligNAtor*, which is described next. *RaligNAtor*'s description follows subsequently.

This software is available as open source under the GNU General Public License Version 3.

2 Database preprocessing with *sufconstruct*

sufconstruct preprocesses a sequence database generating an index to be searched with *RaligNAtor* using algorithm *LESAAlign* or *LGSlinkAlign*. In summary, this procedure consists of reading the target database in FASTA format, mapping the sequences of the database to an alphabet consisting e.g. of characters A, C, G, and U, computing the required index structures according to the desired search algorithm, and saving the structures to files on disk. All this is performed smoothly, where the user only needs to set a few options. An overview of all possible options is given in Table 2.1 and their detailed description is given below.

2.1 Preprocessing options

- `<file>`
  `<file>` is the path and name of the FASTA file for which the is index is to be constructed. The file may contain one or more sequences and all are selected for index construction. Note that index-based search in the forward and reverse complement sequences only requires the construction of a single index.
- `-alph <file>`
  `-alph` takes as parameter the path and name of the text file specifying an alphabet. The sequences' characters are mapped to this alphabet and the sequences are then said to be alphabetically transformed. The index is constructed for the alphabetically transformed sequences. This option also allows for alphabet reduction (see below). Note that the used alphabet will also be used to map pattern characters when the constructed index is searched with *RaligNAtor*.

  Each line in the file specifies a class of characters of the alphabet. These must be ASCII printable characters, i.e. they must have character code between 32 and 127. A class of characters can be of three types:

  – Non-matching characters of the target sequence: specifies characters that can occur in the target sequence but *cannot* match any pattern character. This is useful for cases in which stretches of the target sequence are unknown, commonly rep-

<file>	Load FASTA file
`-alph <file>`	Use alphabet defined in file
`-dna`	Use DNA alphabet {A, C, G, T} and IUPAC wildcards (default)
`-rna`	Use RNA alphabet {A, C, G, U} and IUPAC wildcards
`-lesa`	Construct index for LESAAlign (tables suf and lcp)
`-lgslink`	Construct index for LGSlinkAlign and LESAAlign (tables suf, lcp, and suf^-1)
`-s <index>`	Save constructed structures to given index name
`-x`	Do not save alphabetically transformed sequence
`-c`	Output constructed structures to screen
`-t <file>`	Output constructed structures to text file
`-time`	Display elapsed times

Table 2.1: Overview of options of program *sufconstruct*.

resented by sequences of Ns. There can be only one such character class, specified in one line beginning with symbol `!`. We emphasize that this class does not do any transformation of pattern characters. E.g.

```
!BbNnRrYySsWwKkMmDdHhVv
```

All characters used in this example that occur in the target sequence cause mismatches to any pattern character. However, these characters can be used with a different behavior in the pattern; see the following characters classes.

– Matching characters: a set of characters, whose members are not distinguished between each other, mapping pattern characters to match the same set of characters in the target sequence. In other words, characters (of both the pattern and the target sequence) belonging to one such class are transformed to a single symbol. Hence, this character class can be used for alphabet reduction. Such a character class is specified in one line with a simple list of the member characters. E.g.

```
Aa
```

The class above indicates that `A` and `a` are not distinguished between each other. Another didactic example is

```
AaM
```

This class allows `M` to be used in the pattern, even if it belongs to *non-matching characters of the target sequence*. `M` will be able to match `A`s and `a`s of the target sequence, but it will not match `M`s (if in the target sequence `M` is a non-matching character). We observe that, in the alignments reported by *RaligNAtor*, an alignment column of two matching characters of the same class is marked with symbol `|`, e.g. an alignment of `A` with `a`.

– Wildcards of the patterns: a class of this type specifies a special pattern symbol that can be used to match characters belonging to different *matching character* classes. A typical application is to specify a character e.g. `R` to match `A`s and `G`s in the target sequence, where `A` and `G` belong to two different *matching character* classes. Such a class is specified in one line beginning with a `*`. E.g.

  ```
  *RAG
  ```

  This class defines a wildcard symbol `R`, i.e. the first symbol after `*`, to match `A`s and `G`s in the target sequence. In addition, it will match every character belonging to the classes to which `A` and `G` belong, for instance `a`s and `g`s. Attention: make sure that all characters belonging to this class, except `R`, also belong to a *matching character* class. Otherwise, this wildcard class will not be accepted. We observe that a wildcard character aligned to a *matching character* of its class is annotated with a `+` in the *RaligNAtor* output, as in the following example.

  ```
  Pattern  ...((-..))..(((...)))
           CCCAA-CCUUAAUCCAUARGA
           | ||| |||| ||||| |+||
  Target   CGCAACCCUU-AUC-AAAGGA
           ...((...))-.((.....))
  ```

Naturally, alignments found with *RaligNAtor* show, for each non-gapped position, a single character of the corresponding character class. Each such character is called a *class representative*. By default, the first character different from `!` and `*` of each line is the representative of the class. Another more explicit way to specify the class representative is to end the class definition with a whitespace followed by the desired representative character. As an example, observe that the representative of the class of *non-matching characters of the target sequence* above is `B`. To set it to `N`, define it instead as

```
!BbNnRrYySsWwKkMmDdHhVv N
```

Below is an example of a complete alphabet file.

```
Aa A
Cc C
Gg G
UuTt U
*AG R
*CTU Y
*CA M
*UTG K
*UTA W
```

```
*CG S
*CGUT B
*AGUT D
*ACUT H
*ACG V
*ACGUT N
!NnRrYySsWwKkMmBbDdHhVv N
```

This alphabet file defines four *matching character* classes, whose representatives are `A`, `C`, `G`, and `U`. The class with representative `U`, for example, allows for the use in the pattern of both uppercase and lowercase `U`s and `T`s, such that any of these characters will match both uppercase and lowercase `U`s and `T`s in the target sequence. Because `U` is the class representative, alignments found with *RaligNAtor* will show `U` wherever these characters occur. The file also defines several wildcards that can be used in the pattern, e.g. `R`, to match uppercase and lowercase `A`s and `G`s in the target sequence. Finally, it defines a class of *non-matching characters of the target sequence*. This can contain characters of the previous two classes, e.g. `R`. However, `R`s occurring in the target sequence will cause mismatches, whereas `R` used in the pattern will match uppercase and lowercase `A`s and `G`s in the target sequence. Remember that

  - all characters used to define patterns must belong to a *matching character* and/or *wildcard* class and
  - all characters occurring in the target sequence must belong to a *matching character* or *non-matching character* class.

- `-dna`, `-rna`
  These options allow transforming the input sequences to predefined DNA or RNA alphabets. The alphabets are equal to the alphabet file shown above. The DNA alphabet only differs from the RNA alphabet by having `T` as class representative instead of `U`. If the target sequences contain other characters, one can create a new alphabet in a text file and use it with the option `-alph`.

- `-lesa`
  `-lesa` selects for construction the structures needed for searching the target database with algorithm *LESAAlign*. The structures consist of the suffix array `suf` and the longest common prefix table `lcp`. Note: `suf` and `lcp` are also constructed via option `-lgslink`. Hence, it is not necessary to select option `-lesa` if the database was already processed for search with the *LGSlinkAlign* algorithm.

- `-lgslink`
  `-lgslink` selects for construction the structures needed for searching the target database

with algorithms *LGSlinkAlign* and *LESAAlign*. The structures consist of the suffix array suf, the longest common prefix table lcp, and the inverse suffix array suf^{-1}.

- `-s <index>`
  By using option `-s` along with an index name, each table that is constructed is stored on disk in its own file. The name of each file is `[index name].[table name]`. Additional files are also stored. One file with extension `.alph` stores the alphabet, one with extension `.base` stores basic information about the sequences such as their length, and one with extension `.des` stores the description of each sequence. The sequences and alphabetically transformed sequences are stored in a file with extension `.seq` and `.tseq`, respectively. Note that all the generated files are binary.
- `-x`
  This option prevents *sufconstruct* from saving alphabetically transformed sequences to file. This is useful for saving disk space, but it will require *RaligNAtor* to convert the sequences of the index for each search run.
- `-c`
  `-c` outputs the constructed tables and the corresponding suffixes to screen. This option is only recommended for small databases, say, with sequence length up to 100.
- `-t <file>`
  `-t` works like the option `-c`, but it directs the output to the specified file.
- `-time`
  With this option the elapsed construction time of each table is displayed.

Be aware that the generated files may overwrite existing ones without warning!

2.2 Using *sufconstruct*

We show an example for preprocessing a database for search with algorithm *LGSlinkAlign*. The database, stored in file `Rfam.fas`, consists of sequences obtained from the full alignments of Rfam release 10.1. Below is the program call and its screen output.

```
$ ./sufconstruct /path/to/fasta_file/Rfam.fas -rna -lgslink -s /path/to/save/index/Rfam

Fasta file:          Rfam.fas
Number of sequences: 2756313
Total length:        824991406

Computing suf... done
```

```
Computing lcp... done
Computing suf... done
```

The program execution produces these files:

```
$ ls -goh
total 11.0G
-rw-r-r- 1 68 2012-02-24 16:02 Rfam.alph
-rw-r-r- 1 11M 2012-02-24 16:02 Rfam.base
-rw-r-r- 1 67M 2012-02-24 16:02 Rfam.des
-rw-r-r- 1 790M 2012-02-24 16:08 Rfam.lcp
-rw-r-r- 1 2.1G 2012-02-24 16:08 Rfam.lcpe
-rw-r-r- 1 790M 2012-02-24 16:02 Rfam.seq
-rw-r-r- 1 3.1G 2012-02-24 16:08 Rfam.suf
-rw-r-r- 1 3.1G 2012-02-24 16:08 Rfam.sufinv
-rw-r-r- 1 790M 2012-02-24 16:02 Rfam.tseq
```

3 Searching with *RaligNAtor*

RaligNAtor can search for given sequence-structure patterns in (1) a precomputed index using algorithm *LESAAlign* or *LGSlinkAlign* or (2) directly in a plain FASTA file using algorithm *ScanAlign* or *LScanAlign*. For computing an index, please refer to program *sufconstruct* above. All algorithms deliver the same results, differing for the user only in their running times. For faster index-based and online searches, we recommend using algorithms *LGSlinkAlign* and *LScanAlign*, respectively. An overview of the options of *RaligNAtor* is given in Table 3.1 and are explained in more detail below.

3.1 Search options

- `<data>`
  `<data>` is the path and target FASTA file or the path and prefix name of the files (i.e. file name without extension) storing an index. *RaligNAtor* requires `<data>` to point to a FASTA file in case the user wants to perform an online search with algorithm *ScanAlign* or *LScanAlign* (see options `-scan` and `-lscan` below). For index-based searches with algorithm *LESAAlign* or *LGSlinkAlign*, *RaligNAtor* requires `<data>` to point to an index (see options `-lesa` and `-lgslink` below).
- `-alph`
  `-alph` takes as parameter the path and name of the text file specifying an alphabet. See the full description of alphabet files above in the section about *sufconstruct*.
- `-dna, -rna`
  Alphabet option for the respective kind of sequence. See section about *sufconstruct* for details.
- `-pat <file>`
  `-pat` takes as parameter a text file containing one or multiple sequence-structure patterns describing any (branching, non-crossing) RNA secondary structures. Each pattern is specified in three consecutive lines. The first line begins with the symbol > followed by the description of the pattern. Optionally, the description may be followed by pipe symbols | separating these supplemental options:
  `replacement`, `deletion`, `arc-breaking`, `arc-altering`, `arc-removing`: cost of the respective edit operation, being the same whether the operation occurs in the target

<data>	Index name or FASTA file
-alph <file>	Use alphabet defined by file (option applies only to FASTA file)
-dna	Use DNA alphabet {A, C, G, T} and IUPAC wildcards (default)
-rna	Use RNA alphabet {A, C, G, U} and IUPAC wildcards
-pat <file>	Structural pattern(s) to search for
-for	Search in the forward sequence (default)
-rev	Search in the reverse complement sequence. For searching in the forward sequence as well, combine it with -for
-comp <file>	Load base pair complementarity rules from file
-byseq	Sort matches by sequence and matching position
-byscore	Sort matches of the same pattern by descending score
-byscorea	Sort matches of the same pattern by ascending score
-table	Print matches in table format
-no-overlaps	Filter out low-scoring overlapping matches of the same pattern
-silent	Do not output matches
-progress	Show progress message for each ∼5% processed data
Operation costs and thresholds. These do not override parameters set in the patterns file	
-replacement <cost>	Cost of a base mismatch (default = 1)
-deletion <cost>	Cost of base deletion/insertion (default = 1)
-arc-breaking <cost>	Cost of an arc-breaking (default = 1)
-arc-altering <cost>	Cost of an arc-altering (default = 1)
-arc-removing <cost>	Cost of an arc-removing (default = 2)
-cost <x>	Allow edit distance <= x (default = 0)
-indels <x>	Allow number of indels <= x (default = cost / cost of one indel)
Index-based algorithmic variants*	
-lgslink	Uses early-stop acceleration, enhanced suffix array, and generalized suffix links
-lgslink_nof	Variant lgslink with disabled sequence-based filter
-lesa	Uses early-stop acceleration and enhanced suffix array
*lgslink requires tables suf, lcp, and sufinv. lesa requires only suf and lcp.	
Online algorithmic variants	
-scan	Slides a window over the target sequence reusing matrix entries
-lscan	Scanning variant with early-stop acceleration
-alignngl	Aligns globally reporting the best alignment (no pattern matching)
Chaining options	
-global	Perform global chaining
-local	Perform local chaining
-wf <wf>	Apply weight factor > 0.0 to fragments
-maxgap <width>	Allow chain gaps with up to the specified width
-minscore <score>	Report only chains with at least the specified score
-minlen <length>	Report only chains with number of fragments >= length
-top <#>	Report only top # scoring chains of each sequence
-allglobal	Report for each sequence all global chains satisfying above criteria
-show	Show chains in the report
-show2	Print complete sequences and omit all other matching information

Table 3.1: Overview of options of *RaligNAtor*.

sequence or the pattern. The default cost for `arc-removing` is 2 and for all others it is 1.
`cost`: cost (i.e. sequence-structure edit distance) threshold for matches. Its default value is 0.
`indels`: number of allowed indels. Its default value is the cost threshold divided by the cost of an indel, i.e. `cost/deletion`. Note that since `cost` bounds the number of indels that can actually occur in a match, if `indels*deletion>cost` *RaligNAtor* will also automatically set `indels=cost/deletion`.
`weight`: a weight that is assigned to a chain fragment corresponding to a match of the respective pattern. Its default value is the score associated to a match; see match score definition in *RaligNAtor*'s publication.
`startpos`: used to compute the score of local chains, it denotes the expected matching position of the pattern in the searched sequences. It must be specified for none or all patterns. If not specified, the matching position of the patterns is automatically computed in a stacked way, i.e., the matching position is the sum of the length of all patterns defined before it +1.

Supplemental options must be provided between two pipe symbols and its keyword, e.g. *weight*, is followed by the equal sign (=) and a value. We observe that these options can also be provided in the command line call to *RaligNAtor*, overriding the respective option value given in the patterns file.

The second line of the pattern definition contains the sequence information, i.e., a sequence of bases possibly containing ambiguous IUPAC characters. *RaligNAtor* automatically recognizes ambiguous characters and tries to match the corresponding base, e.g. A or G in place of an R. The third line contains the structure information in dot-bracket notation. In this notation, unpaired bases are represented by dots `.` and paired bases are represented by `(` and `)`. Observe that for specifying a completely single stranded pattern it is necessary to provide a sequence of dots.

As an example, a patterns file may contain the following text.

```
>tRNA-pat|replacement=2|deletion=3|arc-removing=5
GSSVVYRURGYYYARYUGGUUARMRCRYYDSVYUBHHAMBCHRDWRRUYRYRGGUUCRAWUCCYDYHNBBNSYR
(((((((..((((........)))).(((((.......))))).....(((((.......)))))))))))).
```

Another example is a file containing multiple patterns as follows.

```
>ires1|cost=2|indels=0
UGAWCUKD
........
>ires2|indels=1|cost=4
```

```
DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH
((((((.......................))))))
>ires3|indels=0|cost=1
VNHUAUUUADNBWUAC
((((....))))....
>ires4|indels=2|cost=3
CARGAYSNVNNNNDGCRKYCCHVHRWNRUCYAG
(.(((((....((((.....))))..))))).)
>ires5|indels=1|cost=3|deletion=2
BHKHDHDSNBHDRGUNSNSNNNWNN
(((...((((......))))..)))
```

- `-for`
  Option for searching in the forward sequences. This option is selected by default.
- `-rev`
  Option for searching in the reverse complement sequences. If used in combination with the option `-for`, search is performed in both the forward and reverse complement sequences, otherwise search is only performed in the reverse complement sequences. Observe that searching in reverse complement sequences of a database does not require computing an index for the reverse complement sequences. *RaligNAtor* handles this by automatically computing the reverse complement of the patterns and by using these patterns for search. The patterns will contain complement characters according to the IUPAC table. This holds for alphabets specified with option `-dna`, `-rna`, or `-alph`. Characters not belonging to the IUPAC table cannot be complemented and remain unchanged. Base pairing rules are also automatically complemented. This means that, given Watson-Crick and wobble pairs, Watson-Crick pairs remain unchanged but accepted pairs derived from wobble (U, G) and (G, U) pairs automatically become (A, C) and (C, A). Note that (A, C) and (C, A) pairs must not be defined using option `-comp` (see below), since these pairs are then allowed when searching the forward sequences.
- `-comp <file>`
  The parameter of option `-comp` is a file specifying complementary bases. A line with two bases, given without any whitespaces or punctuation, implies that matches to the patterns can contain such a base pair. It is not necessary to specify the pairing rule twice. For example, for pairs (C, G) and (G, C) it suffices to provide a line `CG`. Below is a sample file.

  ```
  AU
  CG
  ```

```
GA
GU
```

According to this file, these base pairs are possible: (A, U), (U, A), (C, G), (G, C), (A, G), (G, A), (U, G), (G, U). Note that if the option `-comp` is not used, Watson-Crick base pairs are allowed by default.

- `-byseq`
  With this option matches are reported by sequence and matching position, such that matches at the beginning of a sequence are reported first. Note that with this option matches are not reported during search as they are found, but only once the search in the entire database is completed.

- `-byscore`, `-byscorea`
  With `-byscore` or `-byscorea` matches are sorted in descending or ascending order of their score, respectively. The match score is inversely proportional to the cost associated to a match; see exact score definition in *RaligNAtor*'s publication. Note that since the score for different patterns is not normalized, matches of the same pattern are reported consecutively.

- `-table`
  Option for reporting the matches in a table format, with one match per row.

- `-no-overlaps`
  `-no-overlaps` filters out low-scoring overlapping matches of the same pattern. More precisely, if the starting and ending positions of a matched substring overlap with the starting and ending positions of another matched substring of the same pattern, only the matched substring with a higher score is reported. In the case of a tie, one of the matches is arbitrarily filtered out. *RaligNAtor* checks several times during search for overlapping matches, hence avoiding a memory overflow in the case of highly sensitive patterns. Note that this option used with the different online and index-based search algorithms does not guarantee an identical output of matches. This can occur due to the different order by which matches are found and filtered out.

- `-silent`
  `-silent` disables the output of matches.

- `-progress`
  `-progress` shows a progress message for each $\sim$5% processed data.

- `-replacement`, `-deletion`, `-arc-breaking`, `-arc-altering`, `-arc-removing`
  Options taking each a value that specifies the cost of the respective edit operation, with meaning and default value as detailed above for option `-pat`. A used option holds for all patterns in a patterns file and overrides the respective value specified in that file. To specify different operation costs for each searched pattern, see option `-pat`.

- `-cost`, `-indels`
  Cost threshold and number of allowed indels for matches. As with the edit operation costs provided in the command line, the value given via these options holds for all patterns of a patterns file and override the respective value specified in that file. To specify different cost thresholds and number of allowed indels for each searched pattern, see option `-pat` above.

- `-lgslink`, `-lesa`
  Selects one of the index-based algorithms *LGSlinkAlign* or *LESAAlign*. These algorithms require an index of the target database, which can be generated with the *sufconstruct* tool above.

  Note: since version 1.1 of *RaligNAtor*, *LGSlinkAlign* performs in a first step sequence-based filtering with standard dynamic programming considering only edit operations on single bases, i.e. insertions, deletions, and replacements. In a second step, it considers also edit operations on base pairs. This filtering can considerably speed up search and affects neither sensitivity nor specificity, but the following condition must be fulfilled. If the cost of an insertion operation is set to e.g. 2, then the cost of an arc altering (option `-arc-altering`) and arc removing (option `-arc-removing`) must be set to at least 2 and 4, respectively, since these imply one and two deletions. The user is responsible for this consistency.

- `-lgslink_nof`
  Selects algorithm *LGSlinkAlign* but does not perform sequence-based filtering.

- `-scan`, `-lscan`
  Selects one of the online algorithms *ScanAlign* or *LScanAlign*. These algorithms operate directly on the database provided as FASTA file.

- `-alignngl`
  Aligns globally each sequence-structure pattern and each sequence of the database reporting the best alignment and the respective sequence-structure edit distance.

We remark that matches are reported on the standard output channel (stdout), whereas additional information such as set costs and thresholds is redirected to the standard error channel (stderr).

3.2 Chaining options

The following options allow to chain matches of the different patterns specified in one patterns file. A chain of matches is a sequence of non-overlapping matches (where each match is then called a chain *fragment*) such that the order of the matches in the chain resembles the order of the respective patterns in the the patterns file.

- `-global`
  Option to perform global chaining of matches.
- `-local`
  Option to perform local chaining of matches.
- `-wf <wf>`
  `-wf` takes as parameter a positive weight factor that is applied to all chain fragments. For instance, if a chain fragment of a pattern has score 2, a weight factor of 10 implies that the chain fragment will have score 20.
- `-maxgap <width>`
  `-maxgap` takes as parameter the maximum distance (i.e. number of bases) allowed between chain fragments.
- `-minscore <score>`
  Report only chains with at least the specified score.
- `-minlen <len>`
  Report only chains with at least the specified number of chain fragments.
- `-top <#>`
  Report only top # scoring chains. If this option is not used, all chains are reported.
- `-allglobal`
  Guarantees that all global chains are reported without discarding any chains with the same score.
- `-show`
  Show chain fragments and their coordinates (i.e. start and end matching position and score) in the chaining report.
- `-show2`
  Print complete sequences for which at least one chain was found and omit all other matching information. A sequence is only printed once. Sequences are printed in their order of occurrence in the database.

We note that chains are reported in descending order of their chain score.

3.3 Using *RaligNAtor*

As an example, we used *RaligNAtor* to search for five patterns derived from the consensus structure of the Rfam family *Cripavirus internal ribosome entry site* (Acc.: RF00458). The patterns, called ires1, ires2, ires3, ires4, and ires5, are shown above in the description of option `-pat`. Here, we stored these patterns in a file called `ires.pat`. The searched database

contained sequences obtained from the full alignments of Rfam 10.1. To search using algorithm *LGSlinkAlign*, we preprocessed this database with *sufconstruct* generating an index called `Rfam`. The allowed base pairs were (A, U), (U, A), (C, G), (G, C), (G, U), and (U, G), which were specified in a text file and used with the option `-comp`. We also set *RaligNAtor* to report global chains of matches with minimum length 5 by using the option `-minlen`. Due to the large number of expected matches for single patterns, we used option `-silent` to prevent matches from being printed out but used option `-show` to print out the resulting chains.

The command call to *RaligNAtor* and the screen output are as follows.

```
$ ./RaligNAtor/path/to/index/Rfam10 -pat /path/to/patterns_file/ires.pat
-comp /path/to/comp_file/rna.comp -lgslink -silent -global -minlen 5 -show

!Number of sequences: 2756313
!Total length:        824991406


!Searching for pattern ires1 in the forward sequence(s)...
Cost threshold (edist) = 2
Max. allowed indels = 0
Min./Max. match length = 8 / 8
Max. match score = 8
Costs: Replacement = 1
Deletion = 1
Arc-breaking = 1
Arc-altering = 1
Arc-removing = 2


Time: 160822.0290 ms
Number of matches: 16033351


!Searching for pattern ires2 in the forward sequence(s)...
Cost threshold (edist) = 4
Max. allowed indels = 1
Min./Max. match length = 35 / 37
Max. match score = 48
Costs: Replacement = 1
Deletion = 1
Arc-breaking = 1
Arc-altering = 1
Arc-removing = 2


Time: 3607395.4620 ms
Number of matches: 8950417


!Searching for pattern ires3 in the forward sequence(s)...
Cost threshold (edist) = 1
Max. allowed indels = 0
Min./Max. match length = 16 / 16
Max. match score = 24
Costs: Replacement = 1
Deletion = 1
Arc-breaking = 1
Arc-altering = 1
Arc-removing = 2


Time: 96774.9180 ms
Number of matches: 1052


!Searching for pattern ires4 in the forward sequence(s)...
Cost threshold (edist) = 3
Max. allowed indels = 2
Min./Max. match length = 31 / 35
```

```
Max. match score = 53
Costs: Replacement = 1
Deletion = 1
Arc-breaking = 1
Arc-altering = 1
Arc-removing = 2


Time: 871779.0860 ms
Number of matches: 112


!Searching for pattern ires5 in the forward sequence(s)...
Cost threshold (edist) = 3
Max. allowed indels = 1
Min./Max. match length = 24 / 26
Max. match score = 39
Costs: Replacement = 1
Deletion = 2
Arc-breaking = 1
Arc-altering = 1
Arc-removing = 2


Time: 798519.5760 ms
Number of matches: 1222639


Total number of matches: 26207571


!Chaining matches... done
Time: 13660.1450 ms


![sequence] [chain score] [chain length] [strand]
>AB183472.1/62866484 171 5 f
0 7 10 18 8
8 43 19 54 47
44 59 79 95 24
60 92 99 132 53
93 117 147 172 39
........ ((((((.........................)))))) ((((....)))).... (.(((((....((((.....))))..)))))..) (((...((((......))))..)))
UGAWCUKD DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNNNDGCRKYCCHVHRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|||+||++ ++++++++++++++++++++++++ ++++++++ +++||||||++++||| ||+||+++++++||+++||+++++++||+|| ++++++++++++||++++++++++
UGAUCUGA UAGAAGUAAGAAAAAUUCCUAGUUAUAUA-UAUUUUUA AGUUAUUUAGCUUUAC CAGGAUGGGGUGCAGCGUUCCUGCAAUAUCCAG CCUUGUAGUUUUAGUGGACUUUAGG
........ ((((((.....................-..)))))) ((((....)))).... (.(((((....((((.....))))..)))))..) (((...((((......))))..)))
>AB017037.1/62866484 171 5 +
0 7 10 18 8
8 43 19 54 47
44 59 79 95 24
60 92 99 132 53
93 117 147 172 39
........ ((((((.........................)))))) ((((....)))).... (.(((((....((((.....))))..)))))..) (((...((((......))))..)))
UGAWCUKD DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNNNDGCRKYCCHVHRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|||+||++ ++++++++++++++++++++++++ ++++++++ +++||||||++++||| ||+||+++++++||+++||+++++++||+|| ++++++++++++||++++++++++
UGAUCUGA UAGAAGUAAGAAAAAUUCCUAGUUAUAUA-UAUUUUUA AGUUAUUUAGCUUUAC CAGGAUGGGGUGCAGCGUUCCUGCAAUAUCCAG CCUUGUAGUUUUAGUGGACUUUAGG
........ ((((((.....................-..)))))) ((((....)))).... (.(((((....((((.....))))..)))))..) (((...((((......))))..)))
>AF218039.1/60286228 171 5 +
0 7 10 18 8
8 43 19 55 48
44 59 80 96 24
60 92 100 133 53
93 117 149 173 38
........ ((((((.........................)))))) ((((....)))).... (.(((((....((((.....))))..)))))..) (((...((((......))))..)))
UGAWCUKD DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNNNDGCRKYCCHVHRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|||+||++ ++++++++++++++++++++++++|++++++++ +++||||||++++||| ||+||+++++++||+++||+++++++||+|| ++++++++++++||++++ +++++
UGAUCUUG UUGUAAAUACAAUUUUGAGAGGUUAAUAAAUUACAA AGCUAUUUAGCUUUAC CAGGAUGCCUAGUGGCAGCCCCACAAUAUCCAG UUUUUCAGAUUAGGUAGUC-GAAAA
........ ((((((.........................)))))) ((((....)))).... (.(((((....((((.....))))..)))))..) (((....(((......)))-..)))
>AF014388.1/60786278 170 5 +
0 7 10 18 8
8 43 19 55 48
44 59 80 96 24
60 92 100 133 52
93 117 150 174 38
........ ((((((.......................)))))) ((((....)))).... (.(((((....((((.....))))..)))))..) (((...((((......))))..)))
UGAWCUKD DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNNNDGCRKYCCHVHRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|||+||++ ++++++++++++++++++++++++|++++++++ +++||||||++++||| ||+||+++++++||+++||+++++++||+|| ++ ++++++++++||++++++++++
```

```
UGAUCUUG UUCCUUAUACAAUUUUGAGAGGUUAAUAAGAAGGAA AACUAUUUAGUUUUAC CAGGAUGCCUAUUGGCAGCCCCAUAAUAUCCAG UU-AUAUGAUUAGGUUGUCAUUUAG
........ ((((((.......................)))))) ((((....)))).... (.(((((.....(((.....)))...)))))) ((-...((((......))))...))
>AF014388.1/60786278 170 5 +
0 7 10 18 8
8 43 19 55 48
44 59 80 96 24
60 92 100 133 52
93 117 149 174 38
........ ((((((.......................)))))) ((((....)))).... (.(((((....((((.....))))..)))))) (((...((((......))))..)))
UGAWCUKD DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNNNDGCRKYCCHVHRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|||+||++ ++++++++++++++++++++++++++|++++++++ +++||||||++++||| ||+||+++++++||+++||+++++++||+|| ++++++++++++||++++++++++
UGAUCUUG UUCCUUAUACAAUUUUGAGAGGUUAAUAAGAAGGAA AACUAUUUAGUUUUAC CAGGAUGCCUAUUGGCAGCCCCAUAAUAUCCAG CUUAUAUGAUUAGGUUGUCAUUUAG
........ ((((((.......................)))))) ((((....)))).... (.(((((.....(((.....)))...)))))) ((....((((......))))...))
>AB006531.1/60036204 170 5 +
0 7 10 18 8
8 43 20 56 47
44 59 82 98 24
60 92 102 135 53
93 117 150 175 38
........ ((((((.......................)))))) ((((....)))).... (.(((((....((((.....))))..)))))) (((...((((......))))..)))
UGAWCUKD DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNNNDGCRKYCCHVHRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|||+||++ ++++++++ ++++++++++++++++++|++++++++ +++||||||++++||| ||+||+++++++||+++||+++++++||+|| ++++++++++++||++++++++++
UGAUCUUA AAAAUUAGGUUAAAUUUCGAGGUUAAAAAUAGUUUU GUAUAUUUAUACUUAC CAAGAUGGACCGGAGCAGCCCUCCAAUAUCUAG GCUCAAACAUUAAGUGGUGUUGUGC
........ ((((((.......................)))))) ((((....)))).... (.(((((....((((.....))))..)))))) ((....((((......))))...))
>EU680971.1/184383 169 5 +
0 7 10 18 8
8 43 19 54 47
44 59 80 96 24
60 92 100 133 51
93 117 147 172 39
........ ((((((.......................)))))) ((((....)))).... (.(((((....((-((.....))))..)))))) (((...((((......))))..)))
UGAWCUKD DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNNN-DGCRKYCCHVHRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|||+||++ ++++++++++++++++++++ ++++|++++++++ +++||||||++++||| ||+||+ ++++++ +||+++||++++++||+|| ++++++++++++||++++++++++
UGAUCUUU AUCGGGACAUGCAAAUGCAAGG-ACAAAACUCCGAU GGAUAUUUAUCCUUAC CAGGAU-CAGCUCAGGCAGCCCCGAAAAAUCCAG CUUCGAAGAGAAGGUGCUCUAGAAG
........ ((((((.................-......)))))) ((((....)))).... (.((((-....((.((.....)))))...)))).) (((...((((......))))..)))
>AF183905.1/56475848 168 5 +
0 7 10 18 8
8 43 20 55 47
44 59 81 97 24
60 92 101 136 50
93 117 151 176 39
........ ((((((.......................)))))) ((((....)))).... (.(((((....(-(((.....)))-)..)))))) (((...((((......))))..)))
UGAWCUKD DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNN-NDGCRKYCCHV-HRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|||+||++ ++++++++++++++++++++++++++|+ ++++++ +++||||||++++||| ||+||+++++++ ++||+++||++ +++++||+|| ++++++++++++||++++++++++
UGAUCUUG UGCGGAGGCAAAAUUUGCACAGUAUAAAA-UCUGCA ACCUAUUUAGGUUUAC CAAGAUCGGUGGAUAGCAGCCCUAUCAAUAUCUAG UUUAGAAGAUUAGGUAGUCUCUAAA
........ ((((((.......................-)))))) ((((....)))).... (.((((.....(.(((.....))).)...)))).) (((...((((......))))..)))
>EF517515.1/55125714 168 5 +
0 7 10 18 8
8 43 20 56 47
44 59 82 98 24
60 92 102 137 50
93 117 152 177 39
........ ((((((.......................)))))) ((((....)))).... (.(((((....(-(((.....)))-)..)))))) (((...((((......))))..)))
UGAWCUKD DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNN-NDGCRKYCCHV-HRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|||+||++ ++++++++++++++++++++ ++++++|++++++++ +++||||||++++||| ||+||+++++++ ++||+++||++ +++++||+|| ++++++++++++||++++++++++
UGAUCUUG UGUGGAGGCAAAAAUUUGCACAGUAUAAAAUCUGCA ACCUAUUUAGGUUUAC CAAGAUCGGUGGAUAGCAGCCCUAUCAAUAUCUAG UUUAGAAGAUUAGGUAGUCUCUAAA
........ ((((((.......................)))))) ((((....)))).... (.((((.....(.(((.....))).)...)))).) (((...((((......))))..)))
>DQ288865.1/58026001 168 5 +
0 7 10 18 8
8 43 20 56 48
44 59 81 97 24
60 92 101 134 52
93 117 149 173 36
........ ((((((.......................)))))) ((((....)))).... (.(((((....((((.....))))..)))))) (((...((((......))))..)))
UGAWCUKD DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNNNDGCRKYCCHVHRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|||+||++ ++++++++++++++++++++++++++|++++++++ +++||||||++++||| ||+||+++++++||+++||+++++++||+|| + ++++++++++||++++++++++
UGAACUUG UCUCUCAACAAAAAGCCACCGACAUUAAGAGAGAGA CCCUAUUUAGGGUUAC CAGGAUCUGCAACAGCAUUCCUGUAUCAUCCAG GG-UGAGGAUUGAGUUGACCUCAUC
........ ((((((.......................)))))) ((((....)))).... (.(((((.....((((.....))))...)))))) ((-...((.(......).))...))
>EF517520.1/55135715 167 5 +
0 7 10 18 8
8 43 19 56 46
44 59 82 98 24
60 92 102 137 50
93 117 152 177 39
........ (-((((((.......................)))))) ((((....)))).... (.((((((....(-(((.....)))-)..)))))) (((...((((......))))..)))
UGAWCUKD D-NNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNN-NDGCRKYCCHV-HRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
```

```
|||+||++ + ++++++++++++++++++ +++++++++++++ +++|||||||++++||| ||+||+++++++ ++||+++||++ +++++||+|| ++++++++++++||++++++++++
UGAUCUUG UCGCAGAGGCAAAAAUUUGCACAGUAUAAAAUCUGCA ACCUAUUUAGGUUUAC CAAGAUCGGUGGAUAGCAGCCCUAUCAAUAUCUAG UUUAGAAGAUUAGGUAGUCUCUAAA
........ (.((((((.......................)))))) ((((....)))).... (.((((.....(.(((.....))).)...)))).) (((...((((......)))).)))
>EF517519.1/55125714 167 5 +
0 7 10 18 8
8 43 19 56 46
44 59 82 98 24
60 92 102 137 50
93 117 152 177 39
........ (-((((((.......................)))))) ((((....)))).... (.(((((....(-(((.....)))-)..)))))) (((...((((......)))).)))
UGAWCUKD D-NNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNN-NDGCRKYCCHV-HRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|||+||++ + ++++++++++++++++++ +++++++|++++++++ +++||||||++++||| ||+||++++++++ ++||+++||++ +++++||+|| ++++++++++++||++++++++++
UGAUCUUG UCGCAGAGGCAAAAAUUUGCACAGUAUAAAAUCUGCA ACCUAUUUAGGUUUAC CAAGAUCGGUGGAUAGCAGCCCUAUCAAUAUCUAG UUUAGAAGAUUAGGUAGUCUCUAAA
........ (.((((((.......................)))))) ((((....)))).... (.((((.....(.(((.....))).)...)))).) (((...((((......)))).)))
>EF517521.1/55135715 167 5 +
0 7 10 18 8
8 43 19 56 46
44 59 82 98 24
60 92 102 137 50
93 117 152 177 39
........ (-((((((.......................)))))) ((((....)))).... (.(((((....(-(((.....)))-)..)))))) (((...((((......)))).)))
UGAWCUKD D-NNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNN-NDGCRKYCCHV-HRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|||+||++ + ++++++++++++++++++ +++++++|++++++++ +++||||||++++||| ||+||++++++++ ++||+++||++ +++++||+|| ++++++++++++||++++++++++
UGAUCUUG UCGCAGAGGCAAAAAUUUGCACAGUAUAAAAUCUGCA ACCUAUUUAGGUUUAC CAAGAUCGGUGGAUAGCAGCCCUAUCAAUAUCUAG UUUAGAAGAUUAGGUAGUCUCUAAA
........ (.((((((.......................)))))) ((((....)))).... (.((((.....(.(((.....))).)...)))).) (((...((((......)))).)))
>AF178440.1/59256123 166 5 +
0 7 10 18 8
8 43 31 66 45
44 59 79 95 24
60 92 99 132 52
93 117 148 172 37
........ ((((((.......................)))))) ((((....)))).... (.(((((....((((.....))))..)))))) (((...((((......)))).)))
UGAWCUKD DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNNNDGCRKYCCHVHRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|||+||++ +++++++++++++++ ++++++++++ ++++++++ +++||||||++++||| ||+||+++++++||+++||+++++++||+|| ++ ++++++++++||+++++++ ++
UGAUCUUG AUUCUGUACAUAAAAGUCGAAAGUAUU-GCUAUAGU GCCUAUUUAGGCAUAC CAGGAUGGCGCGUUGCAGUCCAACAAGAUCCAG UCCUAUACCUCGAGUCGGGUUU-GG
........ (((.((.....................-..)).))) ((((....)))).... (.((((.....((((.....))))..)))).) ((....((((......))))..-))
>AF536531.1/66416834 165 5 +
0 7 5 13 6
8 43 15 50 46
44 59 75 91 24
60 92 95 128 51
93 117 143 168 38
........ ((((((.......................)))))) ((((....)))).... (.(((((....((((.....))))..)))))) (((...((((......)))).)))
UGAWCUKD DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNNNDGCRKYCCHVHRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
| |+ |++ + ++++++++++++++++++++++++|++++++++ +++||||||++++||| ||+||+++++++||+++||+++++++||+|| ++ ++++++++++||++++++++++
UAAUUUGA U-UUAGGUUAUAAUGUUAGGACUAUAAAAAUUAGCU AGUUAUUUAACUUUAC CAAGAUGGCCGUUGGCAGCCCCACGAAAUCUAG CUAUUUUGAUUAGGUGGUCAGAUAG
........ .-((((.......................)))).. ((((....)))).... (.((((......(((.....)))....)))).) (((...((((......)))).)))
>AF022937.1/69357121 162 5 +
0 7 3 11 6
8 43 15 50 44
44 59 68 84 24
60 92 88 121 51
93 117 134 158 37
........ ((((((.......................)))))) ((((....)))).... (.(((((....((((.....))))..)))))) (((...((((......)))).)))
UGAWCUKD DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNNNDGCRKYCCHVHRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|| + |++ + ++ + ++++++++++++++++++ ++++++++ +++||||||++++||| ||+||+++++++||+++||+++++++||+|| ++++++++++++||+++++++ ++
UGUUGUGU U-UGCGCGAUAAAUGCUGACGUGAAAACGUUGCGUA AGCUAUUUAGCUUUAC CAAGACGCCGUCGUGCAGCCCACAAAAGUCUAG GAGCAUACGCUAGGUCGCGUUG-AC
........ (-((((.......................)))).) ((((....)))).... (.((((......(((.....)))....)))).) (.....((((......)))).-.)
>EU282007.1/69357121 162 5 +
0 7 3 11 6
8 43 15 50 44
44 59 68 84 24
60 92 88 121 51
93 117 134 158 37
........ ((((((.......................)))))) ((((....)))).... (.(((((....((((.....))))..)))))) (((...((((......)))).)))
UGAWCUKD DNNNDNDNHNDMWWDYBVNVDNBWHDWADNNNNNNH VNHUAUUUADNBWUAC CARGAYSNVNNNNDGCRKYCCHVHRWNRUCYAG BHKHDHDSNBHDRGUNSNSNNNWNN
|| + |++ + ++ + ++++++++++++++++++ ++++++++ +++||||||++++||| ||+||+++++++||+++||+++++++||+|| ++++++++++++||+++++++ ++
UGUUGUGU U-UGCGCGAUAAAUGCUGACGUGAAAACGUUGCGUA AGCUAUUUAGCUUUAC CAAGACGCCAUCGUGCAGCCCACAAAAGUCUAG GAGCAUACGCUAGGUCGCGUUG-AC
........ (-((((.......................)))).) ((((....)))).... (.((((......(((.....)))....)))).) (.....((((......)))).-.)


Total number of chains: 17
```

Each chain contains the description of the sequence where the chain occurs followed by the chain score, chain length, and matched strand direction (+ for forward or − for reverse). In addition, it contains the fragments' coordinates (i.e. expected or "stacked" start and end matching positions of the fragment, actual start and end matching positions of the fragment, and fragment score) and the matching substring of the fragments along with their sequence-structure alignment to the corresponding patterns.